SYNTACTICALLY GUIDED TEXT GENERATION

A Dissertation
Presented to
The Academic Faculty

By

Yinghao Li

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2020

SYNTACTICALLY GUIDED TEXT GENERATION

Approved by:

Dr. Chao Zhang, Advisor
School of Computational Science and Engineering
Georgia Institute of Technology

Dr. Matthieu Bloch, Co-Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Prof. David V. Anderson
School of Electrical and Computer Engineering
Georgia Institute of Technology

Prof. Ghassan AlRegib
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: April 21, 2020

A mind not to be changed by place or time.

The mind is its own place, and in itself

Can make a heav'n of hell, a hell of heav'n.

John Milton

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Dr. Chao Zhang of the School of Computational Science and Engineering at Georgia Tech, who has devoted a huge amount of time to this project, giving me valuable advices and helping me revise our research article. I would have never completed this thesis without him and without access to the computer cluster he provided. I would also like to thank my co-advisor Dr. Matthieu Bloch of School of Electrical Science and Engineering for co-advising me and encouraging me to carry on when fell frustrated. Moreover, I want to thank Prof. Ghassan AlRegib and Prof. David V. Anderson of Gatech ECE for spending their time reviewing this thesis and my friends in Dr. Chao's group and my family for their support of my research. Most importantly, I sincerely appreciate the effort of all altruistic open-source software developers and researchers on GitHub, Stack Overflow and all other platforms. They deserve the credit for greatly accelerating the development of computer science and bringing the convenience to everyone living in the modern world.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

In recent years, as researchers have achieved breakthrough in generic text generation, increased works have turned their attention to controllable text generation to apply external knowledge to constrain the semantics or syntax of the generated text. In this work, we propose a guided neural text generation framework, which incorporate syntactic guidance to pilot the syntax structure of the output. Two models are included in this framework. The first is a syntax expansion model that expands a high-level constituency parse template to a full-fledged parse using the additional information from the input source text. The second model is for guided text generation. It collects the semantics from the input text and extract the syntactic information from the full-fledged constituency parse, integrating them together to generate sentences that not only have the desired semantics but comply with the target syntax as well. The framework is evaluated on the paraphrasing task with automatic metrics. The results indicate that it outperforms state-of-the-art syntactically controlled text generation models both semantically and syntactically by a large margin.

CHAPTER 1
INTRODUCTION

Text generation, often referred to as natural language generation (NLG) is the task of generating text with the target of simulating human-written text. Considered to be one of the most challenging natural language processing NLP tasks [1, 2], NLG focuses on the construction of a system that produces well-structured texts from some underlying non-linguistic representation of information [3] and includes applications such as weather forecasts generation [4], article or database summarization [5, 6, 7], question answering (QA) [8], dialog generation (chatbot) [9], machine translation (MT) [10], etc. Pioneered by the first sequence-to-sequence (seq2seq) model [11], recent years have witnessed the prevalence of the neural seq2seq framework for text generation [12, 13, 14, 15]. Trained in an end-to-end manner, distributed representations that capture different factors in the input are learnt by a neural encoder and then fed into a neural decoder to generate text under the seq2seq framework.

Despite its success, one limitation of this framework is that the generation process depends on nothing but the input sequence. However, as the input is represented as a whole by one or a sequence of vectors during the information transmission between encoder and decoder in the traditional seq2seq framework, it is impossible to separate different factors (*e.g.*, syntax structure, semantics, sentiment) contained in the input sequence and generate text according to these factors instead of the entire encoded distributed representation, which might be desirable for applications such as paraphrasing [16, 17, 18, 19], text summarization [14, 7, 20], image captioning [21, 22], and dialogue generation [23, 24] to get the ideal generation output.

Under this circumstance, increasing attention has been drawn to developing a text generation model that is able to take external factors into consideration [25, 16, 18, 26, 27].

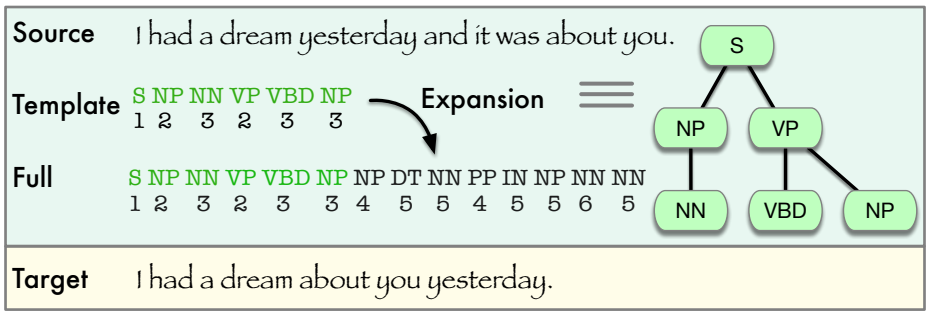


Figure 1.1: The pipeline of our syntactically guided text generation process.

To incorporate syntactic constraints, [18] employs sentence exemplars to represent syntax constraints. [27] uses two latent variables to model semantics and syntax for text reconstruction. [17] also uses deep latent variable models to learn disentangled representations for syntax and semantics. However, in the text generation stage, these prior works do not use any explicit exemplars, but sample from the posterior distribution of the syntax variable to achieve unsupervised style transfer. The most relevant work to ours is conducted by [16], which also uses constituency trees as syntax constraint. To achieve syntactically controlled paraphrase generation, they encode linearized tree sequences along with the source text sequence using long short-term memory (LSTM) networks and inject both the syntax and text hidden representations into the decoding process.

Inspired by the previous research, in this thesis we propose a syntactically guided text generation framework (SGTGen) that incorporates syntactic guidance for text generation. Represented by a full-length constituency parse tree as full guidance or a shallower pruned one as template, as shown in Figure 1.1, this input syntactic guidance can be either explicit or unspecific to provide more freedom to the process. The objective of SGTGen is to generate text that follows the guidance of the input syntactic structure on the bottom of a desired semantics.

SGTGen consists of two models—a syntax expansion model (SynExpan) (Section 3.3) that expands the syntax template into a full-length constituency parse tree, and a guided text generation model (GuiGen) (Section 3.4) that conducts the guided generation from the

source text and the expanded constituency parse. While full-length constituency parses can accurately specify desired syntactic structures, providing such a full-fledged tree beforehand is not always realistic since it is from the target sentence, whereas an arbitrarily chosen one may mismatch the semantics of the input sentence and consequently be unpractical to be directly adopted. Considering this, SynExpan is designed to employ the first few levels of the nodes from the root of a constituency parse tree as a syntax template, and expand it to the full-length constituency tree with additional information from the constituency parse of the input sentence. Combining the Transformer [15], recurrent neural network (RNN) and variational autoencoder (VAE) [28] with von Mish-Fisher (vMF) distribution [29], this model effectively captures the information in both the template and the source input to give a convincing expanded parse, while encouraging variability in generating the full-fledged trees. GuiGen is a Transformer-based seq2seq model that incorporates syntactic guidance whose effectiveness hinges upon the uniquely designed multi-encoder Transformer architecture and the path-attention strategy. We use two Transformer encoders. The semantic encoder is to encode the semantics of the source text, while the syntactic encoder is to encode the syntactic guidance from the constituency parse tree. These two encoders are then connected to one Transformer decoder by a multi-encoder multi-head attention layer so that the information from two encoders are attended simultaneously. Since the syntactic guidance is a linearized constituency parse tree, the path-attention mechanism forces one node to attend only to other tokens located in its path (*i.e.*, its ancestors and descendants) instead of attending to all tokens in the entire sequence equally. In addition, a new scheme for linearizing constituency parse trees is adopted universally in our framework. Unlike traditional schemes that translate the trees into sequences based on recursive representations, our scheme directly expresses the depth information of the nodes by an additional input. Such scheme shortens the encoded sequence by $2/3$, making the incorporation and expansion of the parses more effective.

SGTGen is evaluated on two text generation tasks: paraphrasing and text summariza-

tion (Section 4). Trained and evaluated on the same data, SGTGen consistently outperforms the state-of-the-art baseline models semantically and syntactically. Further investigation also indicates that the order of input text sequence does not affect the generation performance too much, indicating the SGTGen's potential for directly generating plausible sentences from a set of sampled words.

Our main contributions include:

1. a multi-encoder Attention mechanism that allows a Transformer decoder to accept information from multiple Transformer encoders simultaneously, which has been universally applied to all our models;

2. a syntax expansion model SynExpan that integrates VAE, recurrent networks and multi-encoder Transformer to expand partial a syntax template parse into a full-fledged constituency tree tailored for input;

3. a text generation model GuiGen powered by uniquely designed path Attention syntax encoding strategy that jointly attends to the input sentence and the target parse to generate syntax-guided and semantics-preserving text and

4. empirical evaluation that demonstrates the superiority of our model over state-of-the-art controlled text generation methods.

CHAPTER 2

BACKGROUND

2.1 Related Work

2.1.1 Neural Text Generation

Neural text generation based on the encoder-decoder framework has achieved enormous success in various NLP tasks [11, 14, 7, 24, 20]. At the core of this framework is the neural language model (LM) [30], which is often realized by recurrent neural networks (RNN) [31] and its variants such as gated recurrent unit [32, 12] and long short-term memory (LSTM). [33] first introduced the Attention mechanism to recurrent models, which has become prevalent in neural text generation. On the foundation of Attention mechanism, [15] proposed the Transformer architecture for seq2seq neural machine translation (NMT) that eliminated the requirement of the recurrent unit [34, 35]. Since BERT [36] and GPT [37], extra-large Transformer-related models pre-trained on enormous datasets have stolen the limelight by demonstrating extraordinary performance on classification and generation tasks. The Transformer-based generation models are consistently enhanced by more sophisticated masking and pre-training approaches [38, 39, 40, 41, 42, 43] and applied to other generation tasks such as NMT, summarization and even code generation [44, 45].

2.1.2 Constrained Text Generation

Constraint text generation has attracted much attention in recent years. Some works [46, 47, 48, 49, 50] attempt to employ a scalar factor to control or transfer styles in text generation. Specifically, a generator and a discriminator [46] or multiple text decoders [50] corresponding to different styles are learnt and then used for conditional generation. [48] and [47] used formality level parameters and style factor to regularize the output. [49]

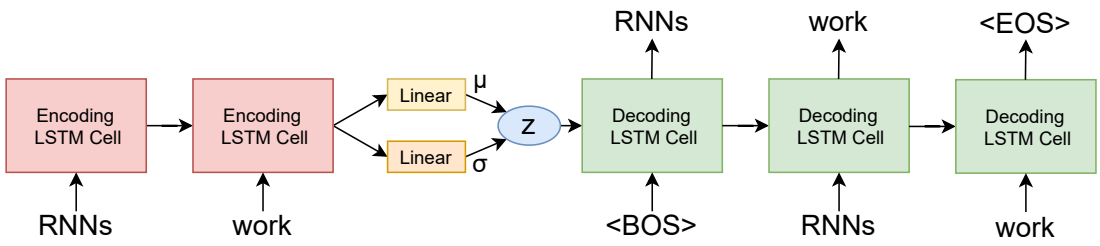


Figure 2.1: The network architecture of RNN-VAE in [13].

changed the sentence sentiment and style by removing the original attribute markers and assigning new words. In addition, attempts have been made to control the generation length for abstractive summarization [51, 52, 53].

In other works, constraints adopted are in a sequential form, *i.e.*, the constraint is a linearized syntax parse tree [16], a sentence [18] or a set of key words [54], which is the case for syntactically constrained text generation. Existing studies on such techniques generally fall into three categories. First, unsupervised syntax control [55, 27, 17] does not require any syntax guidance, but tries to disentangle syntactic and semantic representations for controlled output. For example, [17] uses two latent variables to model the input sentence and its corresponding syntax trees separately. With disentangled representations, they can manipulate the syntax structure in text generation by sampling from the posterior distribution of the syntax variable. Second, several studies control the output syntax structures with sentence exemplars [18, 27]. Given sentence exemplars, they use different latent variables to model semantics and syntax with a deep latent variable model, and control the decoder to generate text that matches the syntax of the given exemplar. Third, there has been work [16] that uses constituency trees as syntax templates, which relates to our work most closely. There has also been work that formulate controlled text generation as filling missing slots of a given template. Therein, the generation process is steered by a learned structural template [25], or by a high-level sentence outline [56].

2.1.3 Variational Autoencoder for Text Generation

Another technique related to our work is VAE [28], which was first introduced to NLP by [13] to generate plausible instances from the interpolation between the latent representations of two sentences encoded by VAE. It uses RNN as encoder and decoder, and Gaussian as the distribution of latent variables, as shown in Fig. 2.1. Afterwards, it is widely applied to different tasks such as text generation [57, 58, 27, 59], sequence matching [60] and sentence classification [61]. [57] regards the entire paraphrase generation training LSTM network as VAE encoder, and uses the last hidden state of the training LSTM decoder to generate the latent variables to train another paraphrase generation network; [58] substitutes the LSTM encoder of [13] by a convolutional neural network (CNN); and [59] uses combines conditional VAE (CVAE) with generative adversarial network (GAN) to generate plausible paraphrases. RNN-VAE is the prevailing model in NLP, but there is also an effort to integrate Transformer with variational autoencoder [62]. To address *posterior collapse* problem [13], Von Mises–Fisher (vMF) distribution is used as the prior and posterior distribution instead of Gaussian distribution in most recent researches [63, 64, 65].

2.2 Related Techniques

2.2.1 RNN, GRU and LSTM

The foundation of the application of neural network in NLP is RNN. Unlike fully connected neural network or CNN where only a finite window of previous input states in a sequence can be considered for conditioning the LM, RNN is capable of conditioning the model on all previous input states, making it suitable for sequential inputs such as sentences or speech signals.

Fig. 2.2 introduces the RNN architecture. $\mathbf{x}^{(t)} \in \mathbb{R}^{d_x}$ is the d_x-dimensional word embedding at time-step t (in the figure $d_x = 4$), $\mathbf{h}^{(t)} \in \mathbb{R}^{d_h}$ is the d_h-dimensional hidden state at time-step t (in the figure $d_h = 5$), $\mathbf{y}^{(t)} \in \mathbb{R}^{d_h}$ is another expression of $\mathbf{h}^{(t)}$, $W_{hh} \in \mathbb{R}^{d_h \times d_h}$

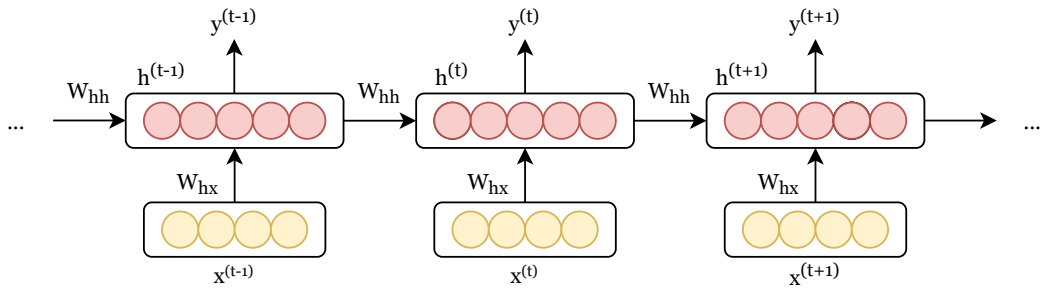


Figure 2.2: The demonstration of RNN architecture.

is the transition matrix between previous and current hidden states and $W_{hx} \in \mathbb{R}^{d_h \times d_x}$ is the affine transition matrix that maps the input $\mathbf{x}^{(t)}$ to the hidden state. $\mathbf{h}^{(t)}$ is recursively calculated by

$$\mathbf{h}^{(t)} = f\left(W_{hh}\mathbf{h}^{(t-1)} + W_{hx}\mathbf{x}^{(t)}\right) \tag{2.1}$$

where $f(\cdot)$ is the non-linear activation function as which the hyperbolic tangent function $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$ or the sigmoid function $\sigma(x) = e^x/(e^x + 1)$ are usually chosen.

Theoretically, RNN can model an input sequence of any length since the number of parameters to be trained does not grow with the increase of sequence length, but in practice it is hard for a hidden state to acquired information from many steps ago due to the vanishing and exploding gradients [66].

To solve the long-term dependency problem, LSTM [67] and GRU [12] have introduced "gates" to control the proportion of information passed from previous states and received from the new input. With these gates, LSTM and GRU are more capable of keeping the information from previous states, thereby making it easier for RNNs to capture long-term dependencies.

A gated recurrent unit is shown in Fig. 2.3. A GRU consists of two gates: an update gate and a reset gate. The update gate is responsible for determining how much of the information consisted in the previous state $\mathbf{h}^{(t-1)}$ should be carried forward to the next state, whereas the reset gate determines the weight of the previous state $\mathbf{h}^{(t-1)}$ when the

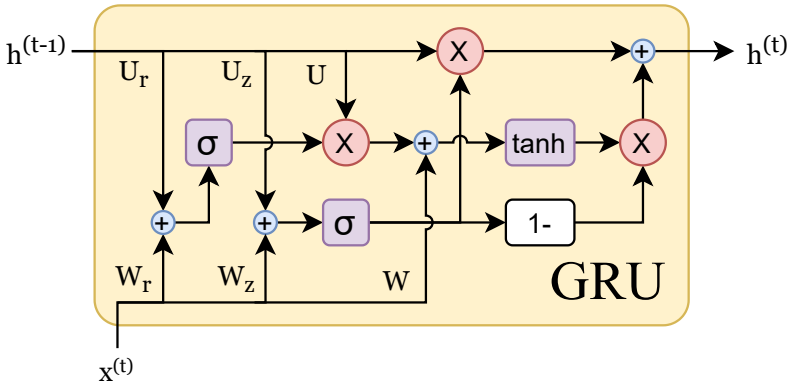


Figure 2.3: The demonstration of a gated recurrent unit.

new memory is calculated. The calculations for the update and reset signal are similar:

$$\mathbf{z}^{(t)} = \sigma\left(W_z\mathbf{x}^{(t)} + U_z\mathbf{h}^{(t-1)}\right), \tag{2.2}$$

$$\mathbf{r}^{(t)} = \sigma\left(W_r\mathbf{x}^{(t)} + U_r\mathbf{h}^{(t-1)}\right), \tag{2.3}$$

where $\mathbf{z}^{(t)}$ is the update signal and $\mathbf{r}^{(t)}$ is the reset signal. The new memory is calculated by

$$\tilde{\mathbf{h}}^{(t)} = \tanh\left(\mathbf{r}^{(t)} \odot U\mathbf{h}^{(t-1)} + W\mathbf{x}^{(t)}\right) \tag{2.4}$$

where $\odot$ denotes pair-wise multiplication. The new hidden state $\mathbf{h}^{(t)}$ is the weighted sum of the new memory $\tilde{\mathbf{h}}^{(t)}$ and the previous hidden state $\mathbf{h}^{(t-1)}$:

$$\mathbf{h}^{(t)} = \left(1 - \mathbf{z}^{(t)}\right) \odot \tilde{\mathbf{h}}^{(t)} + \mathbf{z}^{(t)} \odot \mathbf{h}^{(t-1)}. \tag{2.5}$$

Compared with GRU, LSTM introduces a cell state apart from the original hidden state, and some gates to control the evolution and integration of these states. The gates are calcu-

lated as

$$\mathbf{i}^{(t)} = \sigma\left(W_i\mathbf{x}^{(t)} + U_i\mathbf{h}^{(t-1)}\right), \tag{2.6}$$

$$\mathbf{f}^{(t)} = \sigma\left(W_f\mathbf{x}^{(t)} + U_f\mathbf{h}^{(t-1)}\right), \tag{2.7}$$

$$\mathbf{o}^{(t)} = \sigma\left(W_o\mathbf{x}^{(t)} + U_o\mathbf{h}^{(t-1)}\right), \tag{2.8}$$

where $\mathbf{i}^{(t)}$ is the input gate that determines whether the information in the new input word is useful, $\mathbf{f}^{(t)}$ is the forget gate that assesses whether the information in the past cell state is useful and the output gate $\mathbf{o}^{(t)}$ determines how much information in the new memory will be preserved in the updated hidden state. The cell and hidden states are updated by

$$\tilde{\mathbf{c}}^{(t)} = \tanh\left(W_c\mathbf{x}^{(t)} + U_c\mathbf{h}^{(t-1)}\right), \tag{2.9}$$

$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \odot \tilde{\mathbf{c}}^{(t)}, \tag{2.10}$$

$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \odot \tanh\left(\mathbf{c}^{(t)}\right), \tag{2.11}$$

where $\tilde{\mathbf{c}}^{(t)}$ is the new memory. Compared with GRU, LSTM is more complicated to compute but often achieves slightly better performance.

2.2.2 Attention Mechanisms

Seq2seq was first introduced by [11] to solve the NMT problem and then applied to all kinds of sequence generation problems. Trained end-to-end, it consists of two neural networks—an encoder and a decoder. The encoder takes the model's input sequence and encodes it into a fixed-size hidden vector and the decoder uses it as "seed information" to initialize the network and from which generate the output sequence, as shown in Fig. 2.4. However, traditional seq2seq techniques assign equal importance to all tokens in the input sequence and are unable to consider the different levels of significance when different parts of the output sequences are predicted. For example, while the English sentence "I have an apple."

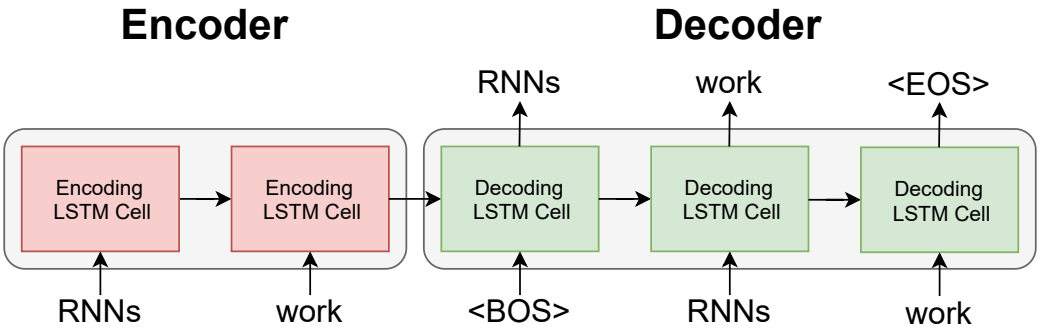


Figure 2.4: The demonstration of the seq2seq model.

is being translated to French "J'ai une pomme.", the prediction of the word "pomme" has nothing to do with "I have" but everything to do with "an apple", which is where the seq2seq models with only one hidden vector to pass information from encoder to decoder fail.

Attention mechanisms have been invented to leverage this observation by providing the decoder network with a look at the entire input sequence at every decoding step and then decide what part of the input sequence are more important to the current prediction. There are two major types of Attentions—Bahdanau Attention (or additive Attention) [33] and Luong Attention (or multiplicative Attention) [68]. In both mechanisms, all hidden states of the encoder are used to compute the next hidden state in the decoder in addition to the decoder's last hidden state and prediction. Let $(\mathbf{h}_e^{(1)}, \ldots, \mathbf{h}_e^{(n)})$ be the hidden vectors representing the input sequence and $(\mathbf{h}_d^{(1)}, \ldots, \mathbf{h}_d^{(t-1)})$ be the decoder hidden states from time-step 1 to $t-1$. The goal is still predicting the token $\hat{y}^{(t)}$ at time-step t.

For Bahdanau Attention, a scalar score is computed for each hidden state from the encoder $\mathbf{h}_e^{(i)}$:

$$s_{t,i} = W \cdot \left(\mathbf{h}_d^{(t-1)} + \mathbf{h}_e^{(i)}\right) \tag{2.12}$$

where $W \in \mathbb{R}^{1\times d}$ is a learnable weight of a feed-forward network and d is the hidden dimension of the network. The scores $(s_{t,1}, \ldots, s_{t,n})$ are then normalized into a weight

vector $\boldsymbol{\alpha}^{(t)} = (\alpha_{t,1}, \ldots, \alpha_{t,n})$ by a softmax layer

$$\alpha_{t,i} = \frac{\exp\left(s_{t,i}\right)}{\sum_{k=1}^{n} \exp\left(s_{t,k}\right)}. \tag{2.13}$$

After that, the context vector $\mathbf{c}^{(t)}$ is calculated as the weighted average of the encoder hidden states:

$$\mathbf{c}^{(t)} = \sum_{i=1}^{n} \alpha_{t,i} \mathbf{h}_e^{(i)}. \tag{2.14}$$

At last, the context vector $\mathbf{c}^{(t)}$ is concatenated together with the embedding of the last prediction $\hat{y}^{(t-1)}$ to calculate the new hidden state $\mathbf{h}_d^{(t)}$, from which the new token is predicted.

In these calculations, $\mathbf{h}_d^{(t-1)}$, $\mathbf{h}_e^{(i)}$ in (2.12) and $\mathbf{h}_e^{(i)}$ in (2.14) can be regarded as *query*, *key* and *value* vectors. Intuitively, (2.12)–(2.14) is to integrate all the *value* vectors together with the weight calculated from *query* and *key* vectors to form a "combined" knowledge represented by the context vector that corresponds to the *query*.

Luong Attention is slightly different from Bahdanau Attention by calculating the new decoder hidden state $\mathbf{c}^{(t)}$ first and concatenating it with the context vector $\mathbf{c}^{(t)}$ instead of the word embedding of the past prediction. In addition, the computation for the score matrix $\mathbf{s}^{(t)}$ is different too. There are three different candidate approaches:

$$\mathbf{s}_i^{(t)} = \mathbf{h}_d^{(t)\mathsf{T}} \mathbf{h}_e^{(i)}, \tag{2.15}$$

$$\mathbf{s}_i^{(t)} = \mathbf{h}_d^{(t)\mathsf{T}} W_{\text{cmb}} \mathbf{h}_e^{(i)}, \tag{2.16}$$

$$\mathbf{s}_i^{(t)} = W \cdot \tanh\left(W_{\text{cmb}} \cdot \left(\mathbf{h}_d^{(t)} + \mathbf{h}_e^{(i)}\right)\right) \tag{2.17}$$

where $W_{\text{cmb}} \in \mathbb{R}^{d \times d}$ is also learnable.

The Attention mechanism allows modeling of dependencies without regard to the distance of tokens in the input or output sequences since it takes all tokens in the sequences globally, and achieved unprecedented performance in various tasks such as NMT, summarization [7] or paraphrase generation [16] and has become an indispensable part of modern

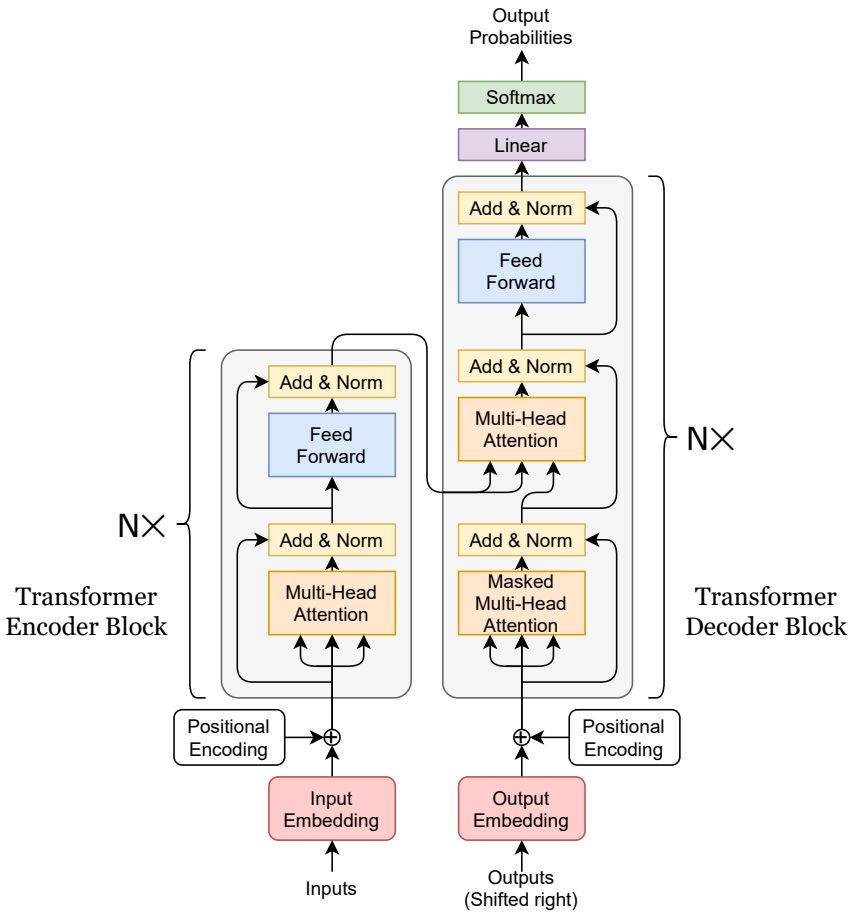


Figure 2.5: The Transformer model architecture [15].

recurrent seq2seq models.

2.2.3 Transformer

However, powerful as RNNs are, another key issue is that their recurrent nature not only prevents them from efficient parallelization but also makes it hard to be expanded in size because of the difficulty of gradient propagation. Such issue becomes prominent as the length of sequences grows. To solve this problem, in [15] the authors proposed Transformer, a novel seq2seq model built on Attention mechanism where the recurrence is excluded. In combination with the residual connections, Transformer can be much larger than recurrent models yet faster to train.

The power of Transformer lies in the Multi-Head Attention, which is the combination

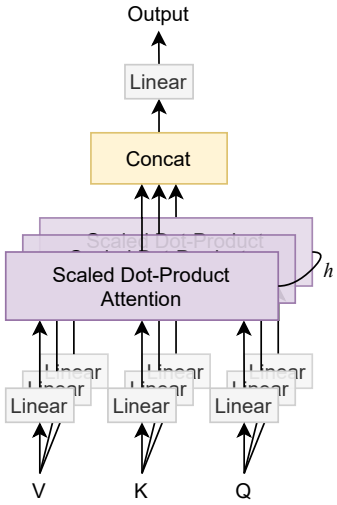


Figure 2.6: The Multi-Head Attention mechanism in [15].

of several parallel Scaled Dot-Product Attentions whose calculation are not unlike those Attentions introduced in Section 2.2.2. Specifically, given a set of combined *query* and *key* vectors $Q, K \in \mathbb{R}^{n \times d_k}$ and *value* vectors $V \in \mathbb{R}^{n \times d_v}$ where d_k is the dimension of *query* and *key* vectors, d_v is the dimension of *value* vector and n is the sequence length, the score matrix is calculated with scaled dot product (same to (2.15) except to the scaling factor) and the context matrix $C \in \mathbb{R}^{n \times d_v}$ is calculated by

$$C = \text{softmax}\left(\frac{QK^\mathsf{T}}{\sqrt{d_k}}\right) \cdot V. \tag{2.18}$$

As shown in Fig. 2.6, for Multi-Head Attention with h heads, (2.18) is repeated h times in parallel with

$$Q_i = QW_Q^{(i)}, \quad K_i = KW_K^{(i)}, \quad V_i = VW_V^{(i)} \tag{2.19}$$

to get C_i with $i \in [1, h]$. $W_Q^{(i)} \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_K^{(i)} \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $W_V^{(i)} \in \mathbb{R}^{d_{\text{model}} \times d_v}$ are trainable matrices with d_{model} the dimension of input and output sequences of a block. Then, the context vectors are concatenated together and multiplied by another matrix $W_O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$. There are two types of Attentions in Transformer: self Attention (The Multi-Head Attention in Encoder and the Masked Multi-Head Attention in Decoder in Fig 2.5)

where *query*, *key* and *value* vectors are all from the previous layer's output, and decoder-encoder Attention (The Multi-Head Attention in Encoder in Fig 2.5) where *key* and *value* vectors are from the output of the encoder.

The positional encodings are injected into the model along with the word embeddings of the input sequence to make fully use of the order of the sequence since the non-recurrent architecture cannot distinguish the relative positions of the tokens in the sequence. They are added to the word embeddings to form a unitary input.

2.2.4 vMF-VAE

The variational autoencoder [28, 69] is a generative model which imposes a prior distribution $P(\mathbf{z})$ on a latent variable $\mathbf{z} \in \mathcal{Z}$ and enforces a regular geometry over the latent space $\mathcal{Z}$ from which a sample can be easily drawn. VAE first encode the input data $\mathbf{x}$ into the latent variable $\mathbf{z}$, and then try to reconstruct $\mathbf{x}$ from it. Mathematically,

$$P(\mathbf{x}) = \int P(\mathbf{x}|\mathbf{z})P(\mathbf{z})d\mathbf{z} = \mathbb{E}_{\mathbf{z}}P(\mathbf{x}|\mathbf{z}) \tag{2.20}$$

is the formula that needs to be optimized. However, directly calculating the integral would be impractical since generating sufficiently many samples from a high-dimensional space is time-consuming. In practice, most $\mathbf{z}$ sampled from the prior distribution $P(\mathbf{z})$ would lead to $P(\mathbf{x}|\mathbf{z}) \approx 0$ hence contributing nothing to the calculation. In light of this, another arbitrary distribution $Q(\mathbf{z}|\mathbf{x})$ is introduced to give a smaller space than the prior to sample $\mathbf{z}$ from and computing $\mathbb{E}_{\mathbf{z}\sim Q}P(\mathbf{x}|\mathbf{z})$ would be relatively easier. Using the arbitrary $Q(\mathbf{z}|\mathbf{x})$ to simulate the real posterior distribution $P(\mathbf{z}|\mathbf{x})$, their Kullback-Leibler divergence (KL divergence) $\mathcal{D}[Q(\mathbf{z}|\mathbf{x})\|P(\mathbf{z}|\mathbf{x})]$ is defined by

$$\mathcal{D}[Q(\mathbf{z}|\mathbf{x})\|P(\mathbf{z}|\mathbf{x})] = \mathbb{E}_{\mathbf{z}\sim Q}[\log Q(\mathbf{z}|\mathbf{x}) - \log P(\mathbf{z}|\mathbf{x})]. \tag{2.21}$$

Applying Bayes' rule to $P(\mathbf{z}|\mathbf{x})$ we get

$$\log P(\mathbf{x}) = \mathcal{D}[Q(\mathbf{z}|\mathbf{x})\|P(\mathbf{z}|\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim Q}[\log P(\mathbf{x}|\mathbf{z})] - \mathcal{D}[Q(\mathbf{z}|\mathbf{x})\|P(\mathbf{z})]. \tag{2.22}$$

As $\mathcal{D}[Q(\mathbf{z}|\mathbf{x})\|P(\mathbf{z}|\mathbf{x})]$ is positive but impossible to compute, we just assume it is small (since $Q(\mathbf{z}|\mathbf{x})$ is representative enough) and ignore it. Therefore, instead of optimizing the log likelihood $\log P(\mathbf{x})$, we optimize the evidence lower bound (ELBO) in the network training process

$$\text{ELBO} = \mathbb{E}_{\mathbf{z}\sim Q}[\log P(\mathbf{x}|\mathbf{z})] - \mathcal{D}[Q(\mathbf{z}|\mathbf{x})\|P(\mathbf{z})] \tag{2.23}$$

where $Q(\mathbf{z}|\mathbf{x})$ is calculated by the network encoder and $P(\mathbf{x}|\mathbf{z})$ is calculated by the decoder. An example is shown in Fig. 2.1.

A conventional choice for the prior and posterior distribution of $\mathbf{z}$ is a Gaussian distribution. In specific, $P(\mathbf{z}) \sim \mathcal{N}(\mathbf{z}|0, I)$ and $Q(\mathbf{z}|\mathbf{x}) \sim \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}, \Sigma)$ where I is the identity matrix and Σ is a diagonal matrix (see Fig. 2.1). However, because of the strong non-linear fitting ability of a neural network, more than often we eventually get $Q(\mathbf{z}|\mathbf{x}) = P(\mathbf{z})$ when the network minimizes the KL divergence $\mathcal{D}[Q(\mathbf{z}|\mathbf{x})\|P(\mathbf{z})]$. Such case is called *KL collapse* [13] or *posterior collapse* [65], where the posterior degrades to the Uniform Gaussian prior, and the encoder's function is lost, leaving the decoder to generate useless output from a bunch of sampled Gaussian noise.

A way to circumvent this problem is to substitute the Gaussian by von Mises-Fisher (vMF) distribution [70, 64, 65]. vMF places a distribution over the $d-1$-dimensional hypersphere defined by mean direction vector $\boldsymbol{\mu} \in \mathbb{R}^d$ with $\|\boldsymbol{\mu}\| = 1$ and a scalar concentration parameter $\kappa \in \mathbb{R}$. For a random vector $\mathbf{x} \in \mathbb{R}^d$ subject to vMF distribution, its probability

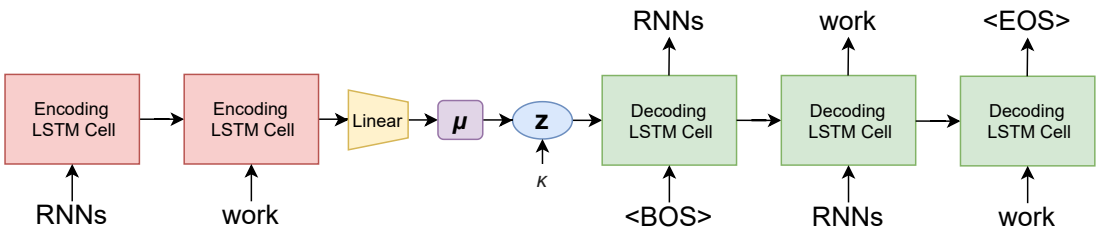


Figure 2.7: A demonstration of the RNN-VAE model when vMF distribution is used.

density function $\mathcal{V}_d(\mathbf{x})$ is defined as

$$\mathcal{V}_d(\mathbf{z}|\boldsymbol{\mu}, \kappa) = C_d(\kappa) \exp\{\kappa \boldsymbol{\mu}^\mathsf{T} \mathbf{x}\}, \tag{2.24}$$

$$C_d(\kappa) = \frac{\kappa^{\frac{d}{2}}}{(2\pi)^{\frac{d}{2}} I_{\frac{d}{2}-1}(\kappa)}, \tag{2.25}$$

$$I_\alpha(\kappa) = \sum_{k=0}^{\infty} \frac{1}{k!\Gamma(k+\alpha+1)} \left(\frac{\kappa}{2}\right)^{2k+\alpha} \tag{2.26}$$

where $\Gamma(\cdot)$ is Gamma function and $I_\alpha(\cdot)$ is the modified Bessel function of the first kind at order $\alpha \in \mathbb{R}$.

The posterior distribution is $Q(\mathbf{z}|\mathbf{x}) \triangleq \mathcal{V}_d(\mathbf{z}|\boldsymbol{\mu}, \kappa)$ where $\boldsymbol{\mu}$ is calculated by the encoder and κ is a constant hyper-parameter, as shown in Fig. 2.7 The prior distribution is a Uniform on hyper-sphere $P_\theta(\mathbf{z}) = \mathcal{V}(\mathbf{0}, 0)$. $\mathbf{z}$ is sampled with acceptance-rejection scheme as introduced by [64].

With these, the KL divergence $\mathcal{D}[Q(\mathbf{z}|\mathbf{x})\|P(\mathbf{z})]$ becomes:

$$\begin{aligned} \mathcal{D}[Q(\mathbf{z}|\mathbf{x})\|P(\mathbf{z})] =& \kappa \frac{I_{\frac{d}{2}}(\kappa)}{I_{\frac{d}{2}-1}(\kappa)} + (\frac{d}{2} - 1)\log\kappa - \frac{d}{2}\log 2\pi \\ & - \log I_{\frac{d}{2}-1}(\kappa) + \frac{d}{2}\log\pi + \log 2 - \log\Gamma(\frac{d}{2}). \end{aligned} \tag{2.27}$$

Since the KL divergence does not depend on $\boldsymbol{\mu}$, the *posterior collapse* problem is solved.

CHAPTER 3

TECHNICAL APPROACH

We explain in detail how our framework SGTGen is built in this chapter. First, we describe the input format of our models, and then introduce the model architectures of guided text generation model GuiGen and the syntax expansion model SynExpan in order.

3.1 Input Format

The input of the whole framework SGTGen consists of three elements—the source text sequence, the source constituency parse sequence and the template constituency parse sequence, where the constituency parses are syntactic representations. An optional input—the constituency parse of the target constituency parse sequence—is also included. The source text sequence is the text that we want to extract semantic information from, *e.g.*, the sentence in the source language in NMT task, the input article to summarize in the summarization task, etc.; the source constituency parse sequence, as indicated by the name, is the linearized constituency parse tree of the source text sequence (which will be referred to as "source parse" in the later discussion); the target constituency parse sequence is the linearized constituency parse tree of the reference (target) text sequence we aim to generate

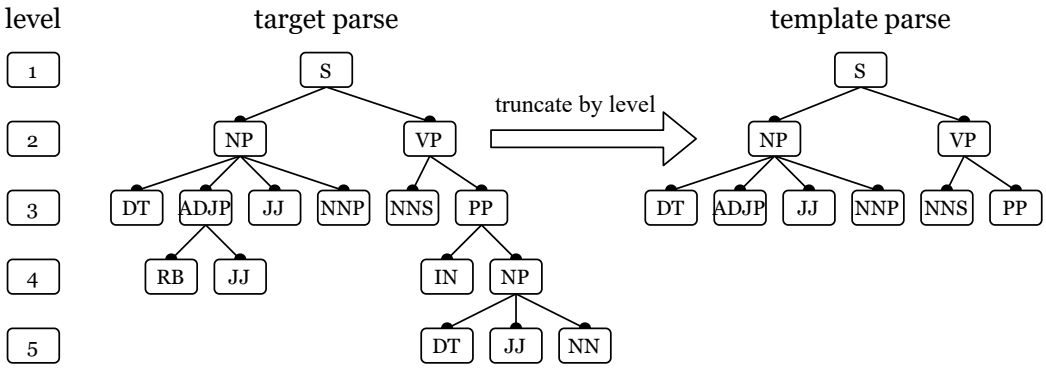


Figure 3.1: An example of the syntax representation.

(which will be referred to as "target parse"); and the template constituency parse sequence is the top-ℓ levels of the target parse (which will be referred to as "template parse"). An example of the source/target parse and the template parse ($\ell = 3$) for the sentence "`The very quick brown fox jumps over the lazy dog`" is shown in Fig. 3.1.[1]

3.1.1 Text Tokenization

In our work, we choose to tokenize the text instances into sub-word units using *byte pair encoding* (BPE) [71, 72] instead of the traditional whole word encoding method. Such strategy not only solves the out-of-vocabulary (OOV) problem of rare words, but also has the ability to model the function of word roots and affixes, making it a stronger and more prevailing tokenization method than the traditional one. In the following discussions, the tokenized text sequence is represented by $\mathfrak{t} = (t^{(1)}, t^{(2)}, \ldots, t^{(M)})$ with $t^{(i)} \in \mathfrak{C}$, where $\mathfrak{C}$ is the set of all sub-word tokens and M the length of the sequence.

3.1.2 Constituency Parse Linearization

As shown in Fig. 3.1, the original format of the constituency parse is a tree, which is inconvenient to be directly treated as the input of a Transformer model. A linearization method is therefore needed to convert the tree structure to a sequential structure. One method called *bracketed format* uses the embedded parentheses to simulate the parent-child relationship in the tree structure [16]. For example, the parse of sentence "I ate an apple." is linearized as "`(S (NP (PRP)) (VP (VBD) (NP (DT) (NN))) (.))`".

In our work, instead, we invent a parse representation differing from the *bracketed format* by expressing the parse using a pair of *node-level* sequences. That is, for the sentence in the previous example, its parse is tokenized as two sequences of the same lengths: "`S NP PRP VP VBD NP DT NN .`" and "`1 2 3 2 3 3 4 4 2`". The node-level

[1] For the meaning of the tags one may refer to the original website `http://www.surdeanu.info/mihai/teaching/ista555-fall13/readings/PennTreebankConstituents.html` or `https://gist.github.com/nlothian/9240750` for supplement.

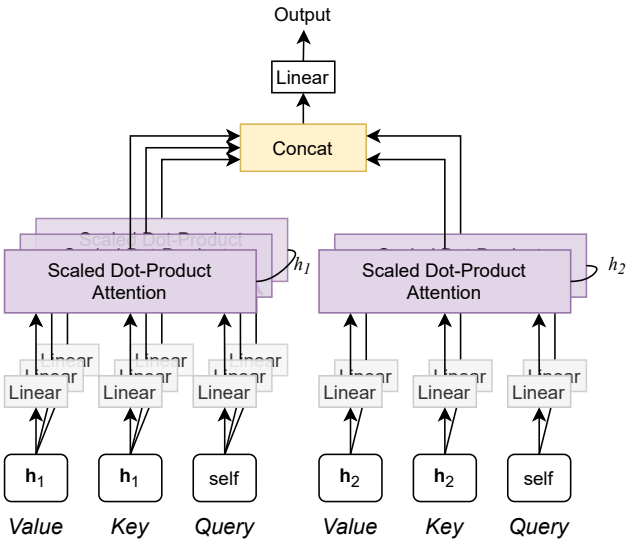


Figure 3.2: A network structure sample of multi-encoder Attention.

representation reduces the parse length to $1/3$ of the bracketed format, significantly decreasing the time consumption for computing as well as making the syntax expansion task much easier. Henceforth, the linearized constituency parse sequences are represented by $\mathbf{s} = (s^{(1)}, s^{(2)}, \ldots, s^{(N)})$ with $s^{(i)} = \{p^{(i)}, l^{(i)}\}$, $p^{(i)} \in \mathfrak{P}$, $l^{(i)} \in \mathfrak{L}$. N is the length of a syntax sequence, $\mathfrak{P}$ is a set holding all constituency parses and $\mathfrak{L}$ is the set of all level tokens.

3.2 Multi-Encoder Attention

In our work, we invent a novel multi-encoder Attention mechanism to connect a Transformer decoder to multiple Transformer encoders so that the decoder can utilize input information from different sources and of different lengths simultaneously to generate desirable sequence accordingly. This mechanism extends the concept of multi-head decoder-encoder Attention by attaching different heads to the output of different Transformer encoders. Fig. 3.2 shows an example of multi-encoder Attention when the number of encoders to attend to is 2. In the figure, $\mathbf{h}_1$ and $\mathbf{h}_2$ are the encoded hidden representation sequences (outputs) from two Transformer encoders, h_1 and h_2 are the number of heads used to attend

to the encoders, and "self" is the output of the former layer. The multi-encoder Attention plays important roles in both SynExpan and GuiGen models.

3.3 Syntax Expansion

The goal of our syntax expansion model SynExpan is to construct a valid full-fledged target parse $\hat{\mathbf{s}}_{\text{tgt}}$ from the template parse $\mathbf{s}_{\text{tmpl}}$ with the additional syntactic information of the parse of the source sentence $\mathbf{s}_{\text{src}}$. It is worth mentioning again that each element in the syntax sequences contains two tokens: $\hat{s}^{(i)}_{\text{tgt}} = \{\hat{p}^{(i)}_{\text{tgt}}, \hat{l}^{(i)}_{\text{tgt}}\}$.

Our syntax expansion model builds a Transformer on top of the RNN-VAE model with vMF distribution to take advantage of both Transformer's strong sequence representation ability and VAE's generation variety. The network structure is shown in Figure 3.3 where the details of Transformer encoder and some layers in decoder are omitted.

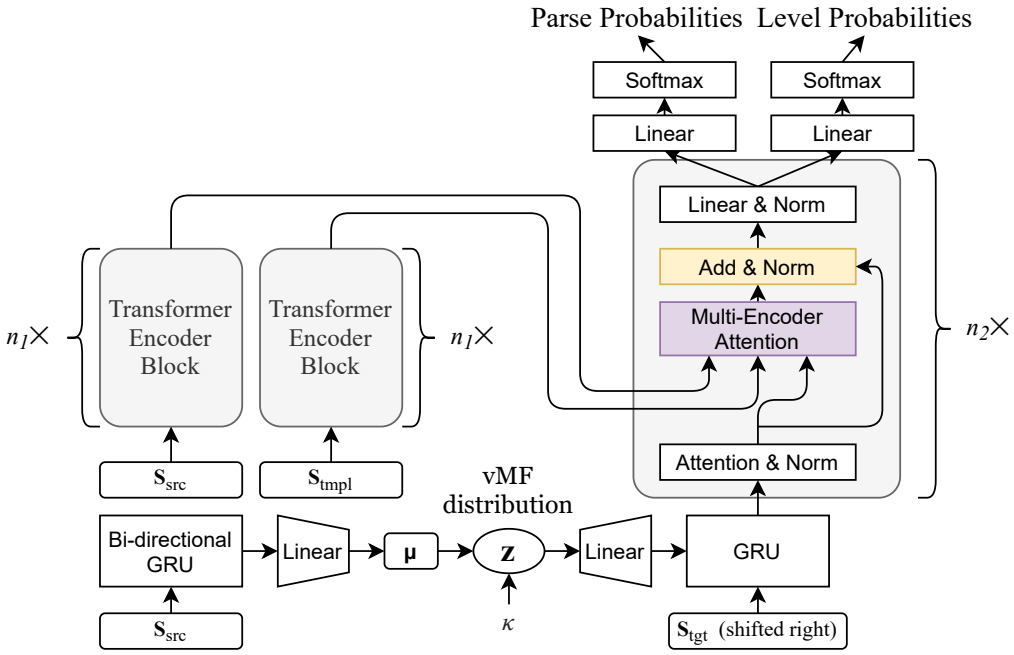


Figure 3.3: Syntax expansion network structure.

3.3.1 Encoding and Decoding

First of all, two standard Transformer encoders are employed to map the source parse $\mathbf{s}_{\text{src}}$ and template parse $\mathbf{s}_{\text{tmpl}}$ to hidden representation sequences $\mathbf{h}_{\text{src}}$ and $\mathbf{h}_{\text{tmpl}}$. At the embedding layer, the parse tokens and level tokens are embedded separately by two linear layers and afterwards added together to form the syntax embedding. Such embedding method is applied universally in our models and will not be specifically mentioned.

In addition, a bi-directional GRU is utilized as the recurrent VAE encoder to fit the distribution $Q(\mathbf{z}|\mathbf{x})$ in (2.23) where $\mathbf{x}$ in our case is $\mathbf{s}_{\text{src}}$. The reason we do not use another Transformer encoder as the VAE encoder is that its output is a sequence of vectors (or in other words, a matrix with flexible height) with clear physical meaning: processed token embedding with each output vector carries some information from not only the input token embedding at the corresponding position but also tokens at other places. However, the parameter $\boldsymbol{\mu}$ that we want to compute is one vector. Using a linear layer for dimension transformation is impractical since the length of the sequence varies, and simply adding all Transformer encoder output vectors violates the intuition.

The decoding network includes two parts—a GRU network decoder, and a modified Transformer decoder that differs from the standard Transformer decoder by substituting the second multi-head Attention layer (the decoder-encoder Attention layer, see Fig. 2.5) with our multi-encoder Attention mechanism. The hidden states of GRU are regarded as the sequence input of the Transformer decoder, and since the recurrence of GRU intrinsically encodes the relative positions, we do not provide external position indicators as [15] does. The sampled $\mathbf{z}$ is not directly used as Transformer decoder input because 1) we want the hidden representation to evolve through the decoding process instead of kept unchanged; 2) a better generation performance is achieved by the current model architecture.

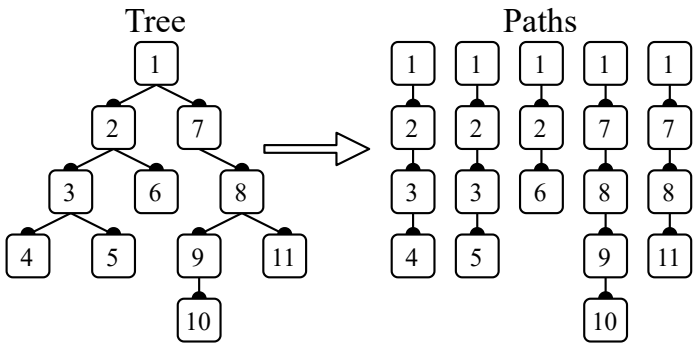


Figure 3.4: An examples of paths.

3.3.2 Training Objective

In this context, the ELBO in (2.23) can be re-written as

$$\text{ELBO} = \mathbb{E}_{\mathbf{z}\sim Q}[\log P(\mathbf{s}_{\text{tgt}}|\mathbf{z})] - \mathcal{D}[Q(\mathbf{z}|\mathbf{s}_{\text{src}})\|P(\mathbf{z})]. \tag{3.1}$$

to optimize ELBO, we need to maximize the probability estimation $\mathbb{E}_{\mathbf{z}\sim Q}[\log P(\mathbf{s}_{\text{tgt}}|\mathbf{z})]$ as well as minimize the KL divergence $\mathcal{D}[Q(\mathbf{z}|\mathbf{s}_{\text{src}})\|P(\mathbf{z})]$. From the VAE perspective, the entire Transformer as well as the GRU decoder can be regarded as the VAE decoder that calculates $P(\mathbf{s}_{\text{tgt}}|\mathbf{z})$. Since the KL divergence is proved in (2.27) to be a constant irrelevant to anything but κ, which is a pre-defined a hyper-parameter, we only need to optimize the prediction probability, or minimize the reconstruction loss, making the training of SynExpan a standard neural network training procedure. However, as a syntax element contains two tokens $s = \{p, t\}$, at each prediction step there are actually two tokens to predict. Therefore, SynExpan is in essence trained in a multi-task fashion.

3.4 Syntactically Guided Text Generation

We proceed to describe our syntactically guided text generation model GuiGen. The purpose of this model is to use external syntax input sequence $\mathbf{s}_{\text{tgt}}$ along with the source text sequence $\mathbf{t}_{\text{src}}$ to guide the generation process, steering the generated text $\mathbf{t}_{\text{tgt}}$ to follow the

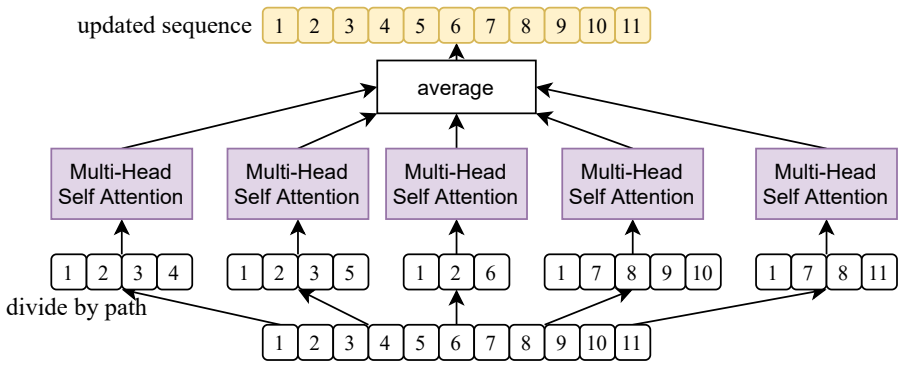


Figure 3.5: Path Attention strategy.

input syntactic structure. The reason we choose the word "guide" instead of "controlled" or "constraint" is that we regard generating the semantically plausible text as the higher priority than syntactically obedient text. Similar to SynExpan, we also use two Transformer encoders to encode syntax sequence and text sequence separately, and a Transformer decoder with multi-encoder Attention layer to generate the wanted text sequence. However, we observe that the situation becomes tricker in the text generation case than syntax expansion. In syntax expansion, those fed into Transformer encoders are all syntax sequences, only varies in length and the maximum tree depth, while in this case those two input sequences comes from two token spaces: $\mathfrak{C}$ and $\{\mathfrak{P}, \mathfrak{L}\}$. Oftentimes because of the provided syntax structure being too specific, the network ultimately learns to map some text tokens to some constituency parses, especially those leaf nodes in the parse tree, which is highly undesirable. To force the network attent to higher-level syntax, which includes more general than specific syntactic guidances, we invent a *path Attention* strategy and modify the syntax encoder accordingly.

In this work, we use the word "path" to describe a route from the root node to a leaf node of a tree. The total number of paths in a tree equals to the number of its leaf nodes. An example of paths is shown in Fig. 3.4. In path Attention strategy, the input of the multi-head Attention is no longer the whole sequence, *e.g.*, tokens at positions 1–11, but multiple (5 in the example) sequences, each containing only the tokens in that path. The outputs of divided sequences are averaged to form an output sequence of the original length. This

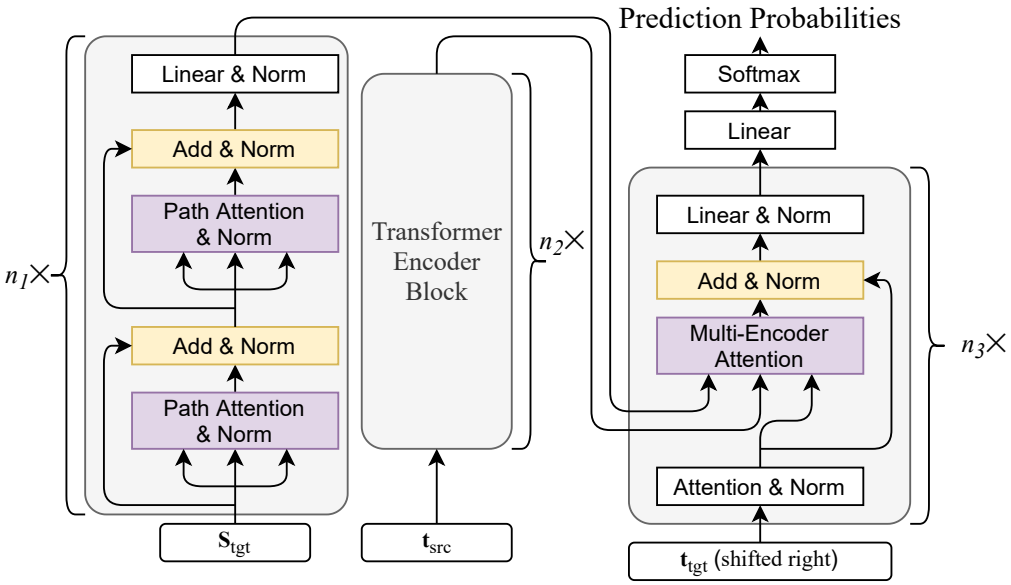


Figure 3.6: Text generation model GuiGen architecture.

process is presented in Fig. 3.5.

In this way, each syntactic node can only attend to other nodes existing on its path and the information transmission between siblings has to go through their parents, forcing parent nodes to carry more information than their children. The Attention process is executed twice in each block so that each node has opportunity to access all other nodes, through root node in the worst case. The model architecture is illustrated in Fig. 3.6

The syntax guidance of GuiGen can either from the constituency parse of the target text $\mathbf{s}_{\text{tgt}}$ or from the output of the syntax expansion model SynExpan $\hat{\mathbf{s}}_{\text{tgt}}$.

CHAPTER 4

EXPERIMENTS

4.1 Experimental Setup

4.1.1 Task and Dataset

Primarily, SGTGen is evaluated on paraphrase generation. Following [16], we first evaluate GuiGen's ability to follow the syntactic guidance by predicting paraphrases with the source text $\mathbf{t}$ and the target parse $\mathbf{s}_{\text{tgt}}$. Then, we assess the performances of SynExpan by predicting text using the constituency tree $\hat{\mathbf{s}}_{\text{tgt}}$ expanded from the constituency tree template $\mathbf{s}_{\text{tmpl}}$. Apart from these, some ablation studies are conducted to prove the usefulness of the elements in those models.

A subset of ParaNMT-50M [73] provided by [18] is used. Text sequences are tokenized into subword units in the form of BPE with the open source python sentencepiece library. [1] The number of total text tokens $|\mathfrak{C}|$ is $16,000$. The constituency parses of the sentences are got from the open-source constituency parsing tool implemented by AllenNLP [74]. [2] Those sentences whose parses contains rare tokens are excluded and all parses are trimmed to 8-level for simplicity, leaving $|\mathfrak{P}| = 74$ and $|\mathfrak{L}| = 12$ including special tokens such as "`<BOS>`" and "`<EOS>`". The dataset is standardized by removing all paraphrase pairs whose text or syntax sequence longer than 50 or with non-ASCII characters. After the preprocessing, $447,536$ paraphrase pairs remain in the processed dataset, in which $402,782$ (90%) are used for training and the rest for validation. Additional 500 and 800 manually annotated high-quality paraphrase pairs are created by [18] for model development and evaluation.

[1] The repository is available at `https://github.com/google/sentencepiece`.
[2] `https://allennlp.org/`

4.1.2 Baselines

We compare our method with SCPN [16] [3] and VAGAE [18] [4], the state-of-the-art controlled paraphrase generation models. Both our mdoel and SCPN take the target sentence's constituency tree as input, except that the constituency tree for SCPN is linearized to the *bracketed representation* while ours to the *node-level* representation. VAGAE uses a sentence as syntax exemplar instead of constituency trees. In addition, we evaluate the paraphrase generated from Transformer without syntactic constraint.

4.1.3 Evaluation Metrics

Following [18], we measure the performance of all the models with three metrics for semantic consistency and one for syntactic consistency. The semantic evaluation metrics include: 1) BLEU [75]; 2) ROUGE [76], including ROUGE-1, ROUGE-2, and ROUGE-L; 3) METEOR [77]. The calculation of BLEU and METEOR is implemented with Python NLTK package [78] [5], while ROUGE is by py-rouge [6]. These metrics calculate the n-grams overlap between generation and reference. While BLEU evaluates ratio of n-grams in the generation that occur in the reference (precision), ROUGE additionally evaluates the ratio of occurrence of reference n-grams in the generation (recall). METEOR uses word embedding distance to evaluate semantics beyond simple overlap of n-grams. To assess the effect of syntactic regulation, the tree-edit distance (TED) of the parse trees of the generated and reference sentences is calculated. TED measures the count of operations, including insertion, rotation and removal, needed for transferring the source tree to the target. However, as TED is in favor of shorter sentences, we sometimes substitute it with normalized version N-TED, *i.e.*, TED divided by the number of nodes in a tree.

[3] The implementation is at `https://github.com/miyyer/scpn`.
[4] `https://github.com/mingdachen/syntactic-template-generation`.
[5] `https://www.nltk.org/`.
[6] `https://github.com/Diego999/py-rouge`

Table 4.1: Text generation results guided by target parse $\mathbf{s}_{\mathrm{tgt}}$ or expanded parse $\hat{\mathbf{s}}_{\mathrm{tgt}}$.

Syntax	Model	B ↑	R-1 ↑	R-2 ↑	R-L ↑	M ↑	T-f ↓	T-8 ↓	T-3 ↓
$\mathbf{s}_{\mathrm{tgt}}$	VGVAE	13.6	44.7	21.0	48.3	24.8	6.7	-	-
	Transformer	15.96	50.11	23.83	49.68	46.84	11.88	11.44	-
	SCPN	23.23	53.21	31.05	57.22	51.91	6.55	6.21	-
	w/o path	38.91	68.01	47.78	70.67	67.26	6.36	**5.88**	-
	GuiGen	**48.03**	**74.53**	**56.05**	**76.65**	**75.02**	**6.23**	5.89	-
$\hat{\mathbf{s}}_{\mathrm{tgt}}$	SCPN	11.83	41.83	20.36	45.63	38.59	8.34	7.77	2.26
	GuiGen	**19.75**	**55.69**	**30.87**	**58.12**	**52.31**	**8.27**	**7.58**	**2.07**

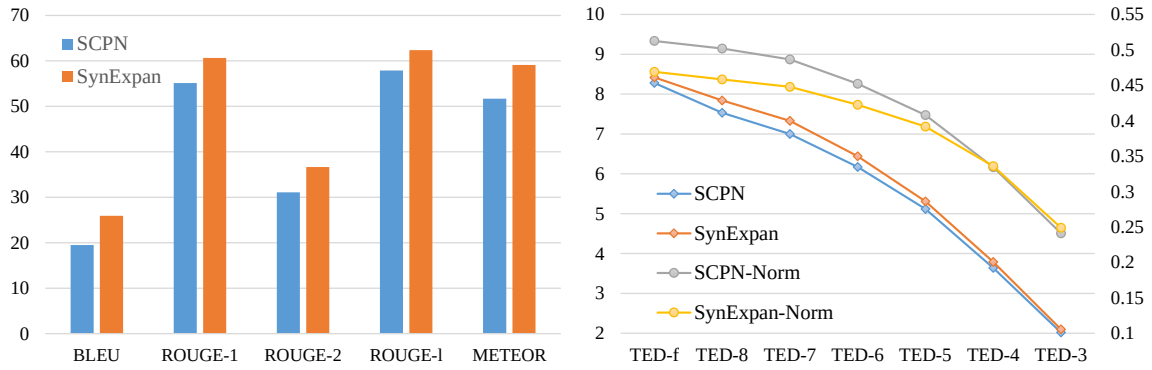
4.2 Quantitative Evaluation

4.2.1 Text Generation with Target Parse

Table 4.1 demonstrates the performance of GuiGen against the baselines when the full constituency parses of target sentences $\mathbf{s}_{\mathrm{tgt}}$ are given as syntactic guidance. In the table, "Transformer" represents the original single-encoder Transformer model trained without the syntax input, "T-f" is the TED calculated on all nodes, "T-8" is the TED with results and references truncated to 8-level, which is the maximum depth of syntax tree used for training. "T-3" is calculated with parses truncated to 3-level, which is the depth of template parse, and "w/o path" is the generation model trained without path Attention strategy. VGVAE's results come from the paper [18]. The results show that our model produces much more better generation results, from both semantic perspective and syntactical perspective, leading to more than doubled BLEU score of the best baseline model SCPN in the mean while achieving smaller TED. Furthermore, the effect of the syntax encoder is examined, and the results indicate that guidance from syntax is also helpful to obtaining desired semantic output.

4.2.2 Syntax Expansion

In this set of experiments, we evaluate the syntax expansion model by predicting text using the constituency tree $\hat{\mathbf{s}}_{\mathrm{tgt}}$ expanded from the template constituency parse $\mathbf{s}_{\mathrm{tmpl}}$. Specifically,



(a) Semantic scores (higher is better).

(b) Syntactic scores (lower is better).

Figure 4.1: Syntax expansion results with GuiGen as text generation model.

we use only the top 3-level of the target constituency tree as a partial syntax template to guide text generation. As it is hard to evaluate the quality of expanded tree directly, we instead evaluate the text generated under the guidance of the expanded parse.

For fair comparison, we first evaluate the generation results of GuiGen and SCPN's text generation network when the syntax guidance is from SCPN's syntax expansion network, showing at the bottom two lines in Table 4.1. It is clear that under the guidance of the same noisy expanded syntactic sequence, GuiGen still has better text generation performance than SCPN. Such being the case, we constantly use GuiGen as the text generation model while comparing the abilities of SynExpan and SCPN's syntax expansion model.

Fig. 4.1 shows the results of two syntax expansion models. Apparently, SynExpan is able to generate more "semantically reasonable" syntactic structures than SCPN. Although it performs slightly worse on TED, as indicated by Fig. 4.1b it is because it tends to give longer predicted structures than SCPN. After the tree edit distance is normalized, SynExpan starts to get better scores, especially when the maximum depth of the parse of generated text increases. To bring in expansion variety, SynExpan compromises a degree of accuracy by introducing vMF-VAE whereas SCPN does not, therefore we can confidently say it is a better syntax expansion model than SCPN's overall.

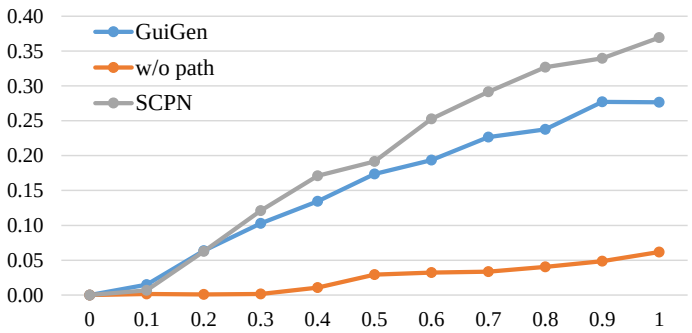


Figure 4.2: the METEOR score decrease rate when a percentage of input text tokens are shuffled.

4.2.3 Path Attention

We further investigate the effect of path Attention strategy. Table 4.1 shows that the GuiGen model trained with path Attention generates sentences with much better semantic scores but similar syntactic scores as the GuiGen without it. This phenomenon is comprehensible since it encourages the model to attend to higher-level syntax, which is less specific than lower-level ones. Trained with path Attention, the network avoids overfitting to the leaf-node parses and learns the real syntax structure rather than a parse-word one-to-one mapping.

Fig. 4.2 tells the same story from another perspective. While GuiGen and SCPN both suffer from the shuffled text input (although GuiGen suffers less due to its concurrent nature), GuiGen w/o path Attention only undergoes minor semantic score decrease even when the sentence is totally shuffled, proving it relies more on the syntax sequence during prediction. Nevertheless, such character may become useful in some special scenarios such as generating sentences from bag of words. The model can be chosen according to the type of application.

4.3 Parameter Study

The concentration parameter κ influences the variability and accuracy of expanded syntax. A suitable κ controls the trad-off between the generation quality and variability. Under

Table 4.2: The influence of the hyper-parameter κ.

κ	BLEU ↑	TED-8 ↓	Self-BLEU
32	15.57	8.55	59.35
128	16.40	8.42	65.22
512	17.27	8.12	77.11
1024	18.13	7.97	81.64

this circumstance, we make a set of comparative trials to find the best κ to use. Table 4.2 lists the performance of syntax expansion model under different κ settings. Self-BLEU measures the generation variability by calculating the BLEU scores for multiple instances generated with different random seeds. Lower Self-BLEU score indicates higher generation variability. As expected, a higher κ value increases the average performance but sacrifices the generation diversity. We eventually choose 768 as the value of κ.

4.4 Qualitative Analysis

4.4.1 Text Generation with Target Parse

Table 4.3 presents some random examples generated with target syntactic parses, from which we can see that the semantic meanings of source sentences are well-preserved while the syntactic style have been transferred based on the parse. Compared with SCPN, text generated from GuiGen has better semantic consistency to the references by 1) using appropriate personal pronouns and 2) avoiding the repeated words problem. However, we observe some defects in our models: the articles and prepositions are tend to be misused (*e.g.*, "a"s are replace by "the" in the 4th example). This may be because the training data tend to change such words, but it can jeopardize the semantics in some situations. Another interesting point is the word "fbi" becomes "fbl" in the first example, which is clearly a disadvantage of using BPE.

Table 4.3: Examples generated with target parse $\mathbf{s}_{\mathrm{tgt}}$.

src: but it looked like it had been raided by the fbi at the time when i saw it. **tgt:** when i saw it, it looked like it had been raided by the fbi. **SCPN:** when i saw him, he looked as he was seen by the fbl. **SGTGen:** when i saw it, it seemed like he had been attacked by the fbl.
src: your child wasn't hurt by me. **tgt:** i did not harm your child. **SCPN:** you didn't hurt my baby. **SGTGen:** i didn't hurt your baby.
src: he was released from prison yesterday. **tgt:** they let him out of prison yesterday. **SCPN:** they give him from jail yesterday yesterday. **SGTGen:** they let him out of jail yesterday.
src: if he insults a german, he will take a bullet. **tgt:** he'll get a bullet for insulting a german. **SCPN:** he'll take a smile in burning a german. **SGTGen:** he'll get the bullet by insulting the german.
src: and i have to sit there on my butt and watch those statements. **tgt:** i have to sit there on my ass, watching those statements. **SCPN:** i have to sit here on my ass, watching these guys. **SGTGen:** i need to sit there on my ass, watching these statements.

Table 4.4: Examples generated with expanded parse $\hat{\mathrm{s}}_{\mathrm{tgt}}$.

src: you told me to look for the wolf where his prey can be found. **tgt:** you said i would find a wolf where i can find its prey. **SCPN:** i thought you'd find it if the wolf saw the wolf. **SGTGen:** you said i should look for a wolf where he could find his prey.
src: wounds on the body may easily be healed, but emotional scars do not go away so easily. **tgt:** physical injuries will heal, but it is not so easy for scars on the soul. **SCPN:** it was a lot of the body, but he 'll have no way of the body. **SGTGen:** the wounds on the body can be healed easily, but emotional scars do n't go so easily.
src: also, to help protect your security, your web browser blocked other content from this site. **tgt:** additional content on the site has also been blocked for security protection. **SCPN:** the site of your web has also protected the protection of your web. if it has saved the other security security. **SGTGen:** the protection of the security of your website has also blocked the other content of this site.
src: we need to further strengthen the agency's capacities. **tgt:** the capacity of this office needs to be reinforced even further. **SCPN:** the possibility of the agency is to survive. **SGTGen:** the capacity of the agency needs to be further strengthened.

4.4.2 Text Generation with Expanded Parse

Table 4.4 shows some randomly chosen examples generated with the guidance of expanded parses. In the table, SCPN indicates that the syntax expansion and text generation models are all from the baseline. The predictions still succeed to preserve most of the source sentences' semantics, but sometimes fail to follow the references' syntactic structures due to the imperfection of the expanded parse. The first and forth examples are perfect, personally we think the first result makes even more sense than the reference. In the second case, the source text has not been paraphrased too much since the high-level syntactic architecture of source and reference sentences are similar, and in the third case the subject and object are mixed up. However, in all cases SCPN persistently generates nonsense, accentuating SGTGen's sophistication.

Table 4.5: Generated examples with frequently appeared templates.

NODE: S NP PRP VP MD VP . **LEVEL:** 1 2 3 2 3 3 2
src: he believed his son had died in a terrorist attack. **gen:** he would believe his son was killed in a terrorist attack.
src: she seems to have written a book about driving. **gen:** she must have written a book on the driver.
src: it is hard for me to imagine where they could be hiding it underground. **gen:** i'd be difficult to imagine where they 're hiding him underground .
NODE: S NP PRP VP VBZ NP . **LEVEL:** 1 2 3 2 3 3 2
src: there were 50 bucks' worth of merchandise stolen by a fucker today . **gen:** it's 50 bucks for the kind of thing stolen by a motherfucker.
src: there's an intelligent way to approach marriage. **gen:** it is a smart way to approach the wedding.
src: stealing state secrets was one thing he was framed for by frank. **gen:** it's a part of the theft of state secrets that frank has been framed.

4.4.3 Text Generation with Common Templates

We take a step further and demonstrate our model's ability to generate sentence from templates that most frequently appeared in the dataset. The examples in Table 4.5 shows that the sentences generated with the same constituency template not only has similar high-level syntactic structure, but have the analogical semantics as the sources as well. The semantic coherency proves that our model has the ability to map the same semantics to different syntactic spaces.

CHAPTER 5
CONCLUSION

We have proposed a novel syntactically guided text generation framework SGTGen that generates sentences under the direction of either full-length constituency parses or template parses containing only top-n level of the full parse as the syntactic guidance. A VAE-based syntax expansion model SynExpan is designed to predict a convincing target parse given a template, and a multi-encoder Transformer model with path Attention strategy GuiGen is introduced to preserve the source text's semantics as well as comply with the syntactic guidance. Quantitative and qualitative experiments demonstrate that our framework generates better sentences than the baselines both semantically and syntactically. Such a robust model can play an important role in syntax prediction and pseudo-data construction tasks.

REFERENCES

[1] S. Lu, Y. Zhu, W. Zhang, J. Wang, and Y. Yu, *Neural text generation: Past, present and beyond*, 2018. arXiv: 1803.07133 [cs.CL].

[2] K. G. Murty and S. N. Kabadi, "Some np-complete problems in quadratic and nonlinear programming," *Mathematical Programming*, vol. 39, no. 2, pp. 117–129, 1987.

[3] E. Reiter and R. Dale, "Building applied natural language generation systems," *Nat. Lang. Eng.*, vol. 3, no. 1, pp. 57–87, Mar. 1997.

[4] A. Belz, "Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models," *Nat. Lang. Eng.*, vol. 14, no. 4, pp. 431–455, Oct. 2008.

[5] C. Aone, M. E. Okurowski, and J. Gorlinsky, "Trainable, scalable summarization using robust NLP and machine learning," in *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, Montreal, Quebec, Canada: Association for Computational Linguistics, Aug. 1998, pp. 62–66.

[6] E. Hovy and C.-Y. Lin, "Automated text summarization and the summarist system," in *Proceedings of a Workshop on Held at Baltimore, Maryland: October 13-15, 1998*, ser. TIPSTER '98, Baltimore, Maryland: Association for Computational Linguistics, 1998, pp. 197–214.

[7] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 1073–1083.

[8] E. Brill, J. Lin, M. Banko, S. Dumais, A. Ng, *et al.*, "Data-intensive question answering," in *TREC*, vol. 56, 2001, p. 90.

[9] A. H. Oh and A. I. Rudnicky, "Stochastic language generation for spoken dialogue systems," in *Proceedings of the 2000 ANLP/NAACL Workshop on Conversational Systems - Volume 3*, ser. ANLP/NAACL-ConvSyst '00, Seattle, Washington: Association for Computational Linguistics, 2000, pp. 27–32.

[10] P. F. Brown, J. Cocke, S. A. Della Pietra, V. J. Della Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. S. Roossin, "A statistical approach to machine translation," *Computational Linguistics*, vol. 16, no. 2, pp. 79–85, 1990.

[11] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14, Montreal, Canada: MIT Press, 2014, pp. 3104–3112.

[12] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734.

[13] S. R. Bowman, L. Vilnis, O. Vinyals, A. Dai, R. Jozefowicz, and S. Bengio, "Generating sentences from a continuous space," in *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 10–21.

[14] R. Nallapati, B. Zhou, C. dos Santos, Ç. Gulçehre, and B. Xiang, "Abstractive text summarization using sequence-to-sequence RNNs and beyond," in *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 280–290.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[16] M. Iyyer, J. Wieting, K. Gimpel, and L. Zettlemoyer, "Adversarial example generation with syntactically controlled paraphrase networks," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 1875–1885.

[17] Y. Bao, H. Zhou, S. Huang, L. Li, L. Mou, O. Vechtomova, X.-y. Dai, and J. Chen, "Generating sentences from disentangled syntactic and semantic spaces," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 6008–6019.

[18] M. Chen, Q. Tang, S. Wiseman, and K. Gimpel, "Controllable paraphrase generation with a syntactic exemplar," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 5972–5984.

[19] S. Wang, R. Gupta, N. Chang, and J. Baldridge, "A task in a suit and a tie: Paraphrase generation with semantic augmentation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 7176–7183, 2019.

[20] A. Celikyilmaz, A. Bosselut, X. He, and Y. Choi, "Deep communicating agents for abstractive summarization," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 1662–1675.

[21] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," *CoRR*, vol. abs/1411.4555, 2014. arXiv: 1411.4555.

[22] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," *CoRR*, vol. abs/1502.03044, 2015. arXiv: 1502.03044.

[23] J. Li, W. Monroe, T. Shi, S. Jean, A. Ritter, and D. Jurafsky, "Adversarial learning for neural dialogue generation," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 2157–2169.

[24] I. V. Serban, T. Klinger, G. Tesauro, K. Talamadupula, B. Zhou, Y. Bengio, and A. C. Courville, "Multiresolution recurrent neural networks: An application to dialogue response generation," *CoRR*, vol. abs/1606.00776, 2016. arXiv: 1606.00776.

[25] S. Wiseman, S. Shieber, and A. Rush, "Learning neural templates for text generation," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 3174–3187.

[26] H. Peng, A. Parikh, M. Faruqui, B. Dhingra, and D. Das, "Text generation with exemplar-based adaptive decoding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 2555–2565.

[27] X. Zhang, Y. Yang, S. Yuan, D. Shen, and L. Carin, "Syntax-infused variational autoencoder for text generation," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 2069–2078.

[28] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, 2013. arXiv: 1312.6114 [stat.ML].

[29] A. Banerjee, I. S. Dhillon, J. Ghosh, and S. Sra, "Clustering on the unit hypersphere using von mises-fisher distributions," *J. Mach. Learn. Res.*, vol. 6, pp. 1345–1382, Dec. 2005.

[30] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *J. Mach. Learn. Res.*, vol. 3, pp. 1137–1155, Mar. 2003.

[31] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Eleventh annual conference of the international speech communication association*, 2010.

[32] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling*, 2014. arXiv: 1412.3555 [cs.NE].

[33] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, cite arxiv:1409.0473 Comment: Accepted at ICLR 2015 as oral presentation, 2014.

[34] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer, "Generating wikipedia by summarizing long sequences," *arXiv preprint arXiv:1801.10198*, 2018.

[35] N. Kitaev and D. Klein, "Constituency parsing with a self-attentive encoder," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 2676–2686.

[36] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2018. arXiv: 1810.04805 [cs.CL].

[37] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018.

[38] K. Song, X. Tan, T. Qin, J. Lu, and T.-Y. Liu, *Mass: Masked sequence to sequence pre-training for language generation*, 2019. arXiv: 1905.02450 [cs.CL].

[39] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, *Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension*, 2019. arXiv: 1910.13461 [cs.CL].

[40] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, 2019.

[41] Z. M. Ziegler, L. Melas-Kyriazi, S. Gehrmann, and A. M. Rush, *Encoder-agnostic adaptation for conditional language generation*, 2019. arXiv: 1908.06938 [cs.CL].

[42] L. Wang, W. Zhao, R. Jia, S. Li, and J. Liu, "Denoising based sequence-to-sequence pre-training for text generation," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 4003–4015.

[43] D. Xiao, H. Zhang, Y. Li, Y. Sun, H. Tian, H. Wu, and H. Wang, *Ernie-gen: An enhanced multi-flow pre-training and fine-tuning framework for natural language generation*, 2020. arXiv: 2001.11314 [cs.CL].

[44] S. Edunov, A. Baevski, and M. Auli, "Pre-trained language model representations for language generation," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4052–4059.

[45] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, *Codebert: A pre-trained model for programming and natural languages*, 2020. arXiv: 2002.08155 [cs.CL].

[46] Z. Hu, Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing, "Toward controlled generation of text," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 1587–1596.

[47] J. Ficler and Y. Goldberg, "Controlling linguistic style aspects in neural language generation," in *Proceedings of the Workshop on Stylistic Variation*, Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 94–104.

[48] X. Niu, M. Martindale, and M. Carpuat, "A study of style in machine translation: Controlling the formality of machine translation output," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 2814–2819.

[49] J. Li, R. Jia, H. He, and P. Liang, "Delete, retrieve, generate: A simple approach to sentiment and style transfer," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 1865–1874.

[50] Z. Fu, X. Tan, N. Peng, D. Zhao, and R. Yan, "Style transfer in text: Exploration and evaluation," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[51] Y. Kikuchi, G. Neubig, R. Sasano, H. Takamura, and M. Okumura, "Controlling output length in neural encoder-decoders," in *Proceedings of the 2016 Conference*

on Empirical Methods in Natural Language Processing, Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 1328–1338.

[52] Y. Liu, Z. Luo, and K. Zhu, "Controlling length in abstractive summarization using a convolutional neural network," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 4110–4119.

[53] I. Saito, K. Nishida, K. Nishida, A. Otsuka, H. Asano, J. Tomita, H. Shindo, and Y. Matsumoto, *Length-controllable abstractive summarization by guiding with summary prototype*, 2020. arXiv: 2001.07331 [cs.CL].

[54] C. Li, W. Xu, S. Li, and S. Gao, "Guiding generation for abstractive text summarization based on key information guide network," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 55–60.

[55] M. Chen, Q. Tang, S. Wiseman, and K. Gimpel, "A multi-task approach for disentangling syntax and semantics in sentence representations," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 2453–2464.

[56] S. Subramanian, S. Rajeswar, A. Sordoni, A. Trischler, A. Courville, and C. Pal, "Towards text generation with adversarially learned neural outlines," in *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, ser. NIPS'18, Montréal, Canada: Curran Associates Inc., 2018, pp. 7562–7574.

[57] A. Gupta, A. Agarwal, P. Singh, and P. Rai, *A deep generative framework for paraphrase generation*, 2017. arXiv: 1709.05074 [cs.CL].

[58] S. Semeniuta, A. Severyn, and E. Barth, "A hybrid convolutional variational autoencoder for text generation," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 627–637.

[59] Q. Yang, z. huo, D. Shen, Y. Cheng, W. Wang, G. Wang, and L. Carin, "An end-to-end generative architecture for paraphrase generation," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3130–3140.

[60] J. Choi, T. Kim, and S.-g. Lee, "A cross-sentence latent variable model for semi-supervised text sequence matching," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 4747–4761.

[61] R. Schumann and I. Rehbein, "Active learning via membership query synthesis for semi-supervised sentence classification," in *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 472–481.

[62] D. Liu and G. Liu, "A transformer-based variational autoencoder for sentence generation," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–7.

[63] K. Guu, T. B. Hashimoto, Y. Oren, and P. Liang, "Generating sentences by editing prototypes," *Transactions of the Association for Computational Linguistics*, vol. 6, 437–450, 2018.

[64] T. R. Davidson, L. Falorsi, N. De Cao, T. Kipf, and J. M. Tomczak, "Hyperspherical variational auto-encoders," *34th Conference on Uncertainty in Artificial Intelligence (UAI-18)*, 2018.

[65] J. Xu and G. Durrett, "Spherical latent spaces for stable variational autoencoders," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 4503–4513.

[66] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ser. ICML'13, Atlanta, GA, USA: JMLR.org, 2013, III–1310–III–1318.

[67] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[68] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 1412–1421.

[69] D. J. Rezende, S. Mohamed, and D. Wierstra, *Stochastic backpropagation and approximate inference in deep generative models*, 2014. arXiv: `1401.4082 [stat.ML]`.

[70] N. I. Fisher, T. Lewis, and B. J. Embleton, *Statistical analysis of spherical data*. Cambridge university press, 1993.

[71] P. Gage, "A new algorithm for data compression," *C Users J.*, vol. 12, no. 2, pp. 23–38, Feb. 1994.

[72] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725.

[73] J. Wieting and K. Gimpel, "ParaNMT-50M: Pushing the limits of paraphrastic sentence embeddings with millions of machine translations," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 451–462.

[74] M. Gardner, J. Grus, M. Neumann, O. Tafjord, P. Dasigi, N. F. Liu, M. Peters, M. Schmitz, and L. Zettlemoyer, "AllenNLP: A deep semantic natural language processing platform," in *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 1–6.

[75] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ser. ACL '02, Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, pp. 311–318.

[76] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81.

[77] S. Banerjee and A. Lavie, "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments," in *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, Ann Arbor, Michigan: Association for Computational Linguistics, Jun. 2005, pp. 65–72.

[78] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.